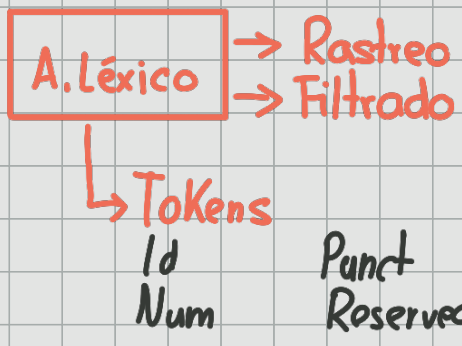
Análisis Léxico

Definición

Verifica símbolos para agruparlos y generar tokens enviados al an. sintáctico.



Los tokens se describen de la forma:

- Expresiones Regulares (RegEx)
- Autómatas Finitos
- Gramáticas Lineales

Componentes Léxicos, Patrones y Lexemas

Es necesario establecer el conjunto de patrones que formarán los tókens para realizar la filtración.

Componente Léxico	Lexema de ejemplo	Descripción informal del patrón.
Palabras Reservadas	if, while, main, do-while	"palabras reservadas"
Literales	3, 5, 9	números
Cadenas	"buaf", "eepy"	símbolos entre comillas
Puntuación	[,], (,), +, -	puntuación
Especiales	#, @, ", <	especiales

Componente Léxico	Lexema de ejemplo	Descripción informal del patrón.
Operadores Generales	*, %, <=, =>	ops. generales
Identificadores	[A,Z]*, [a,z]*, suma, cortar	nombre de variables que inician con una letra seguida de otras y/o dígitas, teniendo como caracter especial válido "_" underscore.

Estos patrones son necesarios para la construcción de un compilador.

Un patrón es una regla que describe al conjunto de lexemas que puede presentar un componente léxico.

No es fácil encontrar la felicidad en nosotros, pero es imposible encontrarla en algún otro lugar.

Agnes Repplier

13/03

Errores Léxicos

No se pueden detectar muchos errores a nivel léxico puesto a que se tiene una visión restringida del código fuente.

```
fi (a = f(x))
```

El analizador léxico no sabe si fi es una variable sin declarar o si es una sentencia if mal escrita

Por lo cual, se delega la responsabilidad de corregir errores a fases siguientes de compilación.

De no ser posible, se emplea el "modo pânico" para intentar corregir el problema:

- → Intercambiar caracteres adyacentes.
- → Borra caracteres extraños
- → Inserta caracteres faltantes
- → Reemplazo de caracteres errôneos.

fi → if
a++b → a+b
buer → where
wher3 → where

RegEx

Expresiones Regulares

Notación para la descripción de patrones, sirven como nombres para un conjunto de cadenas. Tienen como base la Teoría de Lenguajes y de cadenas.

Cadenas y Lenguajes

Conceptos Básicos:

Prefijo de s

Cadena obtenida eliminando n símbolos desde la derecha de la cadena s.

Sufijo de s

Cadena formada suprimiendo uno o más símbolos de la izquierda de la cadena.

Subcadena

Suprime prefijo y sufijo de la cadena.

Prefijo / Sufijo propio de s

Cadena no vacía que sea respectivamente prefijo / sufijo / subcadena de s, tal que se elimine un elemento de s

Subsecuencia de s

Cualquier cadena formada mediante la eliminación de símbolos no necesariamente contiguos de s.

Prefijo

OTORRINOLARINGÓLOGO

OTORRINO

Sufijo

PARANGARICUTIRIMICUARO

-MICUARO, -CUTIRIMICUARO

Subcadena

DESPARANGARICUTIRIMICUIZADO

Pref: DES ; Suf: -izado ∴ parangaricutirimic

Prefijo/Sufijo propio

ELECTROENCEFALOGRAMA

Pref: electro ; Sufi: -grama ; propia: encefalo

Subsecuencia

CARDIOVASCULAR

Subs: o·cular, cardo·lar, ·ardio·lar ...

Definiciones Básicas de Cadenas y Lenguajes

Alfabeto
Conjunto de símbolos utilizados para transmitir información y que forman parte de un lenguaje.
Ejemplo: Alfabeto Castellano {a,b,c}

Lenguaje
Conjunto de cadenas formadas a partir de un alfabeto.
Ejemplo: Idioma Español.

Palabra
Secuencia finita de símbolos de un alfabeto.
Ejemplo: buaf, chetos

Longitud de Cadena
Conjunto de símbolos que componen la cadena.
Ejemplo.

B	U	A	F
1	2	3	4

Esta cadena es de longitud 4.

La longitud de la cadena vacía es 0.

Conjunto Vacío
Conformado únicamente por la cadena vacía. {∅}

Concatenación
Yuxtaposición de dos o más cadenas.

X	Y	XY
buin	ter	**buinter**

Esta operación no es conmutativa. $xy \neq yx$.

Concatenación de Lenguajes
Yuxtaposición de dos o más lenguajes.

X	Y	XY
Español	Griego	**Español** - **Griego**

L^i: Potencia de un alfabeto
Concatenación de L {i} veces.
Ejemplo:

$$s^2 = ss \qquad s^3 = s^2 s = sss \qquad s^4 = s^3 s = ssss$$

Considere que

$$s^0 = \varepsilon$$

Si $A = \{a, b\}$:

$$A^0 = \{\varepsilon\}$$
$$A^1 = \{ab\}$$
$$A^2 = \{abab\}$$
$$A^3 = \{ababab\}$$

Unión de Lenguajes

$$L \cup M = \{x |\ x \epsilon L\ o\ x \epsilon M\}$$

Ejemplo:

L	M	$L \cup M$
$\{\$\}$	$\{a, e, i, o, u\}$	$\{\$, a, e, i, o, u\}$

Cerradura Estrella (Kleene)

$$L^* = L^0 \cup L^1 \cup L^2 \cdots$$

L^* denota cero o más concatenaciones de L. Contiene a la cadena vacía.

Cerradura Positiva

$$L^+ = \{L^0, L^1, L^2, L^3, L^4, \cdots\}$$

L^+ denota una o más concatenaciones de L. No contiene a la cadena vacía.

Ejemplo: Considere los siguientes conjuntos y describa los nuevos que se generan a partir de las expresiones y conceptos vistos anteriormente.

v = {jitomate, berenjena, ajo, lechuga, zanahoria, cebolla, alcachofa}

b = {coca, agua, jugo, tequila, mojito, pulque, kosako, tonayan, aguas locas}

c = {chetos, doritos, sabritas, chachitos, sabritas crema & especias, sabritones, paketaxo}

f = {guayaba, mamey, mango, manzana, chabacano, ciruela, fresas, sandia, uva, pitacaya}

1. $v \cup c$ = {jitomate, berenjena, ajo, lechuga, zanahoria, cebolla, alcachofa,chetos, doritos, sabritas, chachitos, sabritas crema & especias, sabritones, paketaxo}
   La longitud de los elementos de $v \cup c = 1$.
2. $b^2 v$ = {coca agua jitomate, jugo tequila berenjena, pulque kosako ajo, tonayan aguas locas lechuga}
   La longitud de los elementos de $b^2 v = 3$.
3. $c(f^* v)^3$ = {sabritas mango jitomate mango jitomate⋯}
4. $f^3 b \cup c$ = {guayaba guayaba guayaba coca, sabritas crema & especias}

Expresiones Regulares

Nombres para Expresiones Regulares Extensas

A menudo se usan expresiones para identificar y ahorrar espacio en Expresiones Regulares compuestas.

Por ejemplo, una expresión regular como:

$$(1|2|3|4|5|6|7|8|9)\ (1|2|3|4|5|6|7|8|9)^*$$

Se pueden representar como:

$$\text{digito digito}^*$$

donde: digito= {1,2,3,4,5,6,7,8,9}

Ej. Con las expresiones vistas anteriormente, encuentre una expresión regular que reconozca todos los posibles identificadores. Recordando que los identificadores y/o dígitos con una letra seguida de letras y/o dígitos 0 o más veces, teniendo como único carácter especial el underscore.

$$\text{digito}=\{1,2,3,4,5,6,7,8,9\}$$
$$\text{letra}=\{A,B,C,D,\cdots,Z,\ a,b,c,\cdots,z\}$$

Por lo tanto,

$$\text{id=Letras(letra|digito|_)*}$$

Ej. Encuentre una expresión regular para las palabras reservadas if, for, while, do, print, scanf, main.

$$\text{reservada=if,for,while,do,print,scanf,main}$$

Abreviaturas en la notación

- Cero o el operador postfijo ?, significa el caso “0 o 1 en caso de:”. Suele aparecer una vez.

$$r?\ equivale\ a\ r|\varepsilon$$

- Clases de caracteres, la notación $[a\ b\ c]$ donde a, b, c son símbolos del alfabeto. Designan la expresión regular $a|b|c$. Una de todas las posibilidades del conjunto.

$$[A-Z] equivale A|B|C|D|\cdots|Z$$

- Cualquier carácter, para explicar la coincidencia de cualquier símbolo del alfabeto se una un punto.

$$"."*\ \ b\ "."*\ \textit{representa todas las cadenas que tengan una b.}$$

- Cualquier carácter que no este en un conjunto dado, la exclusión de un carácter o cadena se realiza mediante la negación NOT que es representada por la tilde ~.

$$\sim a\ \textit{define a cualquier símbolo que no es a.}$$

- Subexpresiones opcionales, un suceso frecuente es que algunas partes de la expresión regular sean opcionales

$$\text{natural}=[0\text{-}9]^*$$
$$\text{naturalSigno}=(+|\text{-})?\text{natural}$$

Ej. Los comentarios en Pascal se escriben de la siguiente forma:

```
{buenas tardes alumnado}
{buaf}
{mimido}
```

Se debe iniciar con una llave que abre y termina con una de cierre. No se permiten comentarios anidados.
Por lo tanto, $\{\sim(\{|\})*\}$

Elaborar regex que reconozca a todos los enteros con o sin signo como: -3, 1, 4, 70, +8.
Considere no válidas los ceros a la izquierda.

enteroCero [(+|-)?[1-9][0-9]* | 0

Elabore la regex que genere los números reales con y sin signo. Como ejemplo: -3.1416, -4.6, +0.2, 0.5.
Considere inválido tener ceros a la izquierda y/o derecha.

//usando el ej anterior como base.

enteroCero. ([0-9]* [1-9] | 0

se asume -0.0 y +0.0 posibles y válidos

reescrito:

(+|-)?([1-9][0-9]* | 0). ([0-9]* [1-9] | 0)

reales Signo

Genere regex que reconozca las constantes numéricas en lenguaje C.

como
```
int   ganancia = 2500;
float pi = 3.1416;
float buqf = -5'000 000;
      suma = 0
```

(int ? id = enteroCero) | (float id = reales Signo)

Encuentre el regex para palabras reservadas: if, for, while, case, when, printf, do while.

reservedWord = {if, for, while, case, when, printf}

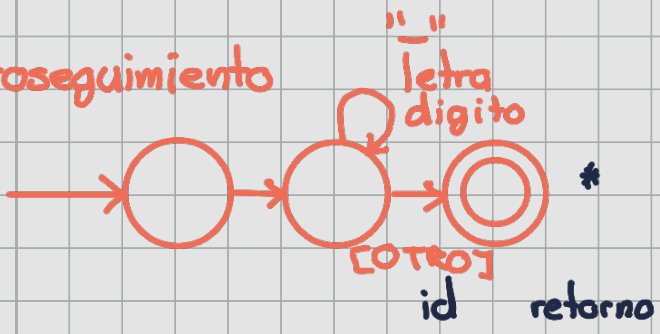
Reg Ex que desarrolle la definición de funciones por parte del usuario.

```
int  buaf ()                    float buef(int n1, int n2, int n3)
int  eepy ()                    char mongo(int tam, int rep,
                                  char simbolo)
```

(int|float|char|void) id([[(int|float|char) id, *](int|float|char) id]])?)

Búsqueda hacia adelante y Retroseguimiento

Considere el siguiente autómata finito para el reconocimiento de identificadores con valor retorno y delimitador



Este autómata no dejará de funcionar hasta que lea algo fuera del conjunto existente. Una vez encontrado ese delimitador, retorna el token resultante a la tabla de tipos.

M I M I D O = 1 0 0 ;

es correcto ya que esos caracts. existen para aceptar la cadena.

---

0 1 | 0 4

Tabla de Transiciones

Es posible identificar una AFD como una estructura de datos y escribir el código genérico relacionado con ésta. Una estructura adecuada para este propósito es la Tabla de Transiciones, que es un arreglo bidimensional formado por estados y caracteres de entrada que expresan los valores de las funciones de transición T.

	Símbolos del alfabeto
Estados S	Estados representando transiciones T(S,C)

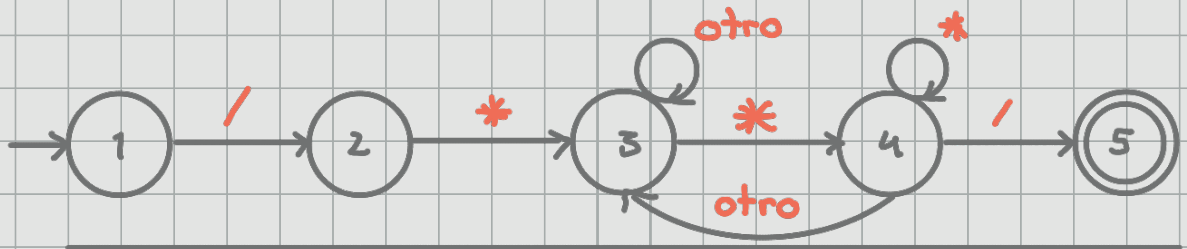
Ej.

AFD pl identificadores

Produce la salida:

EDO \ SIMBOL	LETRA	DÍGITO	"_"	OTRO	Aceptación
1	2	–	–	–	No
2	2	2	2	[3]	No
3	–	–	–	–	Sí

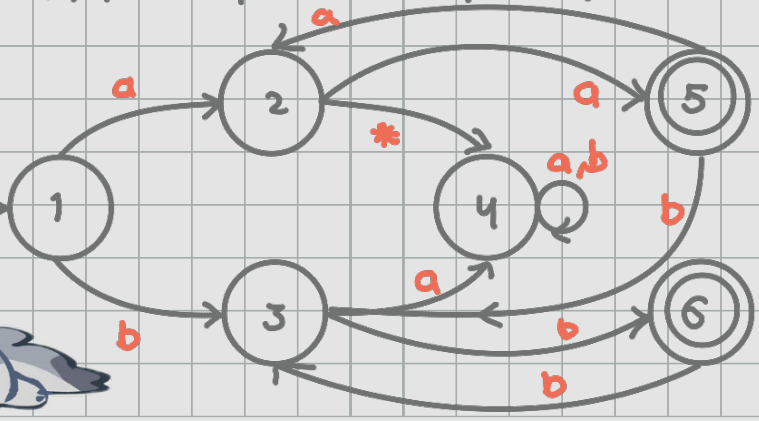
Ej. Dibuja la tabla de transiciones del AFD siguiente.



	/	*	otro	aceptación
1	2	·	·	N
2	·	3	·	N
3	·	4	3	N
4	5	4	3	N
5	·	·	·	Y

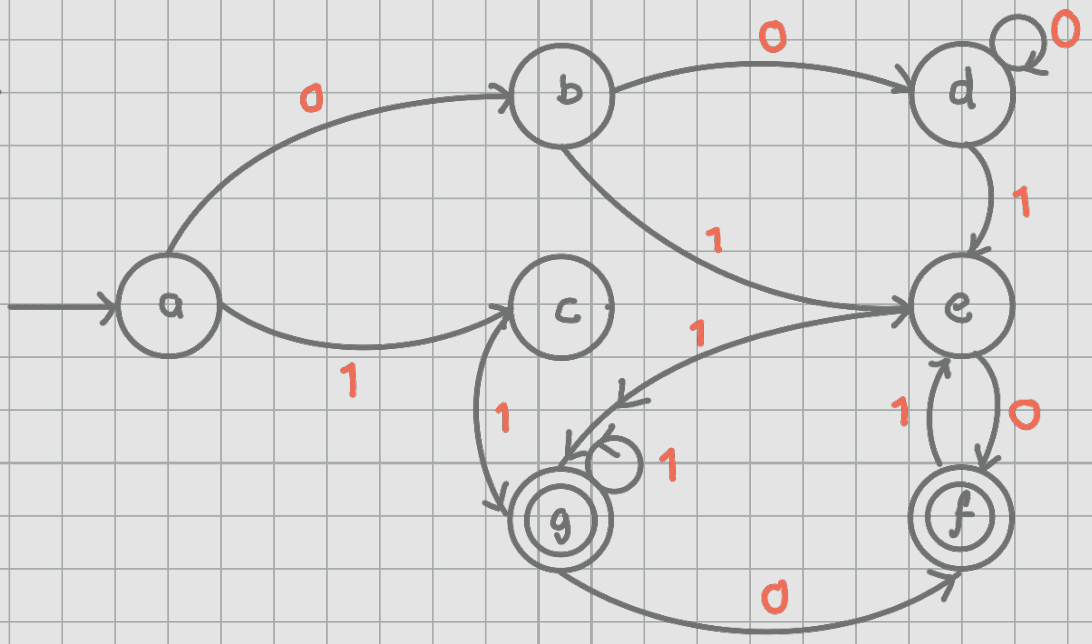
Este autómata reconoce comentarios en lenguaje C.
No delimita el tóKen que se ingresa.

Ej. Construya la tabla de transiciones para el autómata:



	a	b	*	aceptación
1	2	3	·	N
2	5	·	4	N
3	4	6	·	N
4	4	4	·	N
5	2	3	-	S
6	·	3	·	S

Ej.



	1	0	aceptado
a	c	b	N
b	e	d	N
c	g	·	N
d	e	d	N
e	g	F	N
f	e	·	Y
g	g	f	Y

Autómatas Finitos no Determinísticos.

03|04

Un AFN M consta de un alfabeto Σ, un conjunto de estados S y una función de transición:

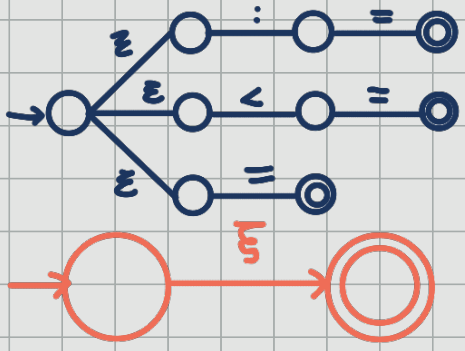
$$T : S_x (U \ \Sigma \cup \{\varepsilon\}) \rightarrow Y(s)$$

así como un estado de inicio S0 y un conjunto de aceptación $A \in S$.

Una transición ξ (ksi) es aquella que puede ocurrir sin consultar la cadena de entrada y sin consumir ningún caracter.

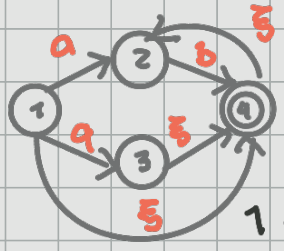
Otra ventaja de las transiciones ksi ξ es que describen completamente una coincidencia de la cadena vacía.

Una ventaja es que permite expresar una selección de alternativas de forma que no involucre combinar estados



Dado el AFND, encuentre el reconocimiento de la cadena

a bb



1 — a → 2 — b → 4 — ξ → 2 — b → 4

1 — a → 3 — ξ → 4 — ξ → 2 — b → 4 — ξ → 2 — b → 4

De reg Ex a AFD

El algoritmo más simple para traducir una regex pasa por una construcción interna en la cual se deriva un AFND de la regex y posteriormente se usa para construir un AFD equivalente.

REGEX ⟶ AFND ⟶ AFD → PROGRAMA

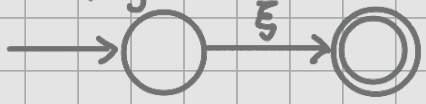
regEx a AFND

La Construcción de Thompson usa transiciones ξ para concatenar las máquinas de cada segmento de la regex, con la finalidad de construir máquinas más complejas.

AFND básicos

- Reconocimiento regex "a"

- Para ξ es



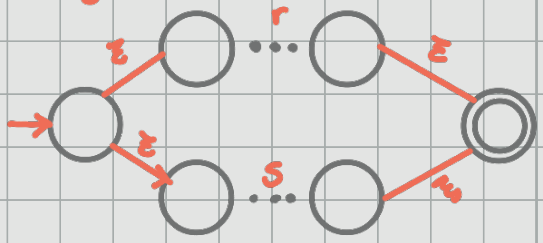
Concatenación / Yuxtaposición

tomando en cuenta el regex "rs" se tiene:

··· indica que pueden existir n transiciones entre 1 y 2

resultando en el autómata

Disyunción



Cerradura *